

Poglejmo, kako reševati čisto prvo domačo nalogo z generatorskimi izrazi. Spominimo se: naloga je videti tako.

Imamo datoteko takšne oblike.

```
11
17
24
30
-1
13
27
33
-1
12
27
34
40
-1
9
-1
8
20
30
31
-1
```

Tule se je prodajalo pet predmetov.

- Ana je za prvi predmet ponudila 11, Berta 17, Ana 25, Berta 30. Številka -1 označuje, da je predmet prodan. Prvega je torej kupila Berta za 30.
- Za drugi predmet je Ana ponudila 13, Berta 27. Prodano Ani za 33.
- Za tretjega Ana 12, Berta 27, Ana 34, Berta 40. Prodano Berti za 34.
- Za četrtega Ana 9. Prodano Ani za 9. (Šlo je samo za star polomljen dežnik, ki pa je imel za Ano neko emocionalno vrednost, ker jo je spominjal na njeno teto po očetovi strani iz Lesničnega.)
- Za petega Ana 8, Berta 20, Ana 30, Berta 31. Berti za 31.

Vidimo torej:

1. Skupno so prodali 5 predmetov.
2. Najvišja dosežena cena je 40.
3. Skupna cena prodanih predmetov je 143 (to je,  $30 + 33 + 34 + 9 + 31$ ).
4. Ana je kupila 2 predmeta, Berta pa 3.
5. Ana je porabila 42, Berta 101.

Napisati nam je program, ki prebere in izračuna vse te stvari.

## 1. Število predmetov

Prešteti moramo, koliko je enic. Najočitnejše je: pokličemo `list`, kot argument damo odprto datoteko. Preštejemo, koliko vrstic je enakih `"-1\n"`.

```
list(open("drazba.txt")).count("-1\n")
```

5

Marsikaj lahko očitamo tej rešitvi. Od tega, da moramo dodajati  $\backslash n$ , do tega, da nas bo zmedel vsak dodaten presledek. Pa še celotno datoteko preberemo in shranimo v seznam.

Za začetek: preberimo *številke* iz datoteke.

```
print([int(v) for v in open("drazba.txt")])
```

[11, 17, 24, 30, -1, 13, 27, 33, -1, 12, 27, 34, 40, -1, 9, -1, 8, 20, 30, 31, -1]

Še vedno beremo v seznam - tega se bomo znebili kasneje. Že to je boljše, saj nas prazni prostor ne moti več. Rešitev, podobna prejšnji vendar varnejša, je torej

```
[int(v) for v in open("drazba.txt")].count(-1)
```

5

V resnici nas zanima samo, ali je številka enaka -1. Pa pripravimo seznam **True**-jev in **False**-ov.

```
print([int(v) == -1 for v in open("drazba.txt")])
```

```
[False, False, False, False, True, False, False, False, True, False, False, False, False, True]
```

Zanima nas, koliko je **True**-jev. To bi lahko (spet) dobili s **count**, lahko pa kar seštejemo elemente tega seznama, saj je **True** toliko kot 1 in **False** toliko kot 0.

```
sum([int(v) == -1 for v in open("drazba.txt")])
```

5

Končno, namesto da sestavljamo seznam, lahko funkciji `sum` podamo kar generator.

```
sum(int(v) == -1 for v in open("drazba.txt"))
```

5

Tak način programiranja nas bo spremljal naslednjih nekaj tednov.

## 2. Najdražji predmet

Naloga pač sprašuje po največji številki. To je še preprosteje kot število predmetov:

```
max(int(v) for v in open("drazba.txt"))
```

40

Za vajo (in prihodnjo rabo) se lotimo še malo drugače: izračunajmo maksimum vseh zadnjih cen predmetov.

Ko smo reševali naloge z datotekami smo (namerno!) odlašali s tem, da bi spoznali in uporabljali seznane. Naučili smo se, da je koristno poznati trenutno in prejšnjo vrstico datoteke. Zdaj vemo za `pairwise`, ki vrne zaporedne pare.

```
from itertools import pairwise
```

```
pairwise(open("drazba.txt"))
```

```
<itertools.pairwise at 0x1084e9ea0>
```

No, ja, `pairwise` je generator, ki generira par za parom. Če hočemo, da jih "izgenerira" in pokaže, jih zložimo v seznam.

```
print(list(pairwise(open("drazba.txt"))))
```

```
[('11\n', '17\n'), ('17\n', '24\n'), ('24\n', '30\n'), ('30\n', '-1\n'), ('-1\n', '13\n'), ('13\n', '30\n'), ('30\n', '33\n'), ('33\n', '40\n'), ('40\n', '9\n'), ('9\n', '31\n')]
```

Zanimajo nas cene v vrsticah, ki jim sledi vrstica -1.

```
[int(prej) for prej, potem in pairwise(open("drazba.txt")) if int(potem) == -1]
```

```
[30, 33, 40, 9, 31]
```

Ker `pairwise` generira pare zaporednih elementov, gremo čeznje z zanko, `for prej, potem in pairwise(open("drazba.txt"))`. Zanimajo nas le tisti pari, pri katerih je drugi element enak -1, saj to pomeni, da je bil izdelek v tem trenutku prodan. Zato dodamo pogoj `if int(potem) == -1`. V seznam pa zlagamo končne cene, torej `int(potem)`.

Najdražji prodani izdelek je potemtakem:

```
max(int(prej) for prej, vrstica in pairwise(open("drazba.txt")) if int(vrstica) == -1)
```

```
40
```

### 3. Skupna cena prodanih predmetov

Ker smo se malo bolj potrudili pri prejšnji točki, nam je zdaj preprosto dobiti skupno ceno prodanih izdelkov. Dobimo jo iz natančno takšnega seznama kot prej, le namesto `max` pokličemo `sum`.

```
sum(int(prej) for prej, vrstica in pairwise(open("drazba.txt")) if int(vrstica) == -1)
```

```
143
```

### 4. Koliko predmetov je kupil kdo

Za vsak izdelek moramo vedeti, koliko ponudb je bil deležen. Zato moramo dobiti številke vrstic, v katerih je prišlo do nakupa - se pravi številke vrstic, ki vsebujejo -1.

```
nakupi = [i for i, v in enumerate(open("drazba.txt")) if int(v) == -1]
```

```
nakupi
```

```
[4, 8, 13, 15, 20]
```

Z `enumerate(open("drazba.txt"))` smo dobili seznam parov (številka vrstice, vrstica). Razpakiramo jih v `i` in `v`. Naredimo seznam tistih `i`-jev (številke vrstic), v katerih je vsebina vrstice enaka `-1`. Zdaj pogledamo te številke.

```
nakupi
```

```
[4, 8, 13, 15, 20]
```

Za prvi izdelek so bile ponudbe štiri (v vrsticah 0, 1, 2, 3; v četrti je bil prodan). Za drugega so bile ponudbe tri (5, 6, 7), za tretjega štiri (9, 10, 11, 12), za četrtega 1 (14) in za petega spet štiri (16, 17, 18, 19). Število ponudb torej dobimo tako, da odštejemo dva zaporedna elementa `nakupi` in od razlike odštejemo še 1. Prvi predmet je poseben, z njim se bomo ukvarjali potem.

Razlike med zaporednima elementoma dobimo preprosto.

```
ponudb = [x - y - 1 for y, x in pairwise(nakupi)]
```

```
ponudb
```

```
[3, 4, 1, 4]
```

Zdaj pa rešimo še prvi element. Preprosto bo: delali se bomo, ko da je pred ničto vrstico - torej v minus prvi vrstici - še ena `-1`. Torej, kot da bi bile `-1`-ke v vrsticah

```
[-1] + nakupi
```

```
[-1, 4, 8, 13, 15, 20]
```

Pa zdaj preštejmo število ponudb:

```
ponudb = [x - y - 1 for y, x in pairwise([-1] + nakupi)]
```

```
ponudb
```

```
[4, 3, 4, 1, 4]
```

Berta je kupila vse predmete s sodim številom ponudb, torej jih je kupila

```
sum(x % 2 == 0 for x in ponudb)
```

```
3
```

Ana pa, seveda, tiste z lihim:

```
sum(x % 2 == 1 for x in ponudb)
```

```
2
```

## 5. Poraba

Cene vseh izdelkov znamo dobiti, to smo počeli že v drugi točki.

```
cene = [int(prej) for prej, vrstica in pairwise(open("drazba.txt")) if int(vrstica) == -1]
```

Zanima nas vsota vseh cen, za katere je bilo število ponudb, recimo, sodo, če nas zanima Bertina poraba.

```
sum(cena for cena, ponudb_zanj in zip(cene, ponudb) if ponudb_zanj % 2 == 0)
```

```
101
```

Z `zip` smo sestavili cene in števila ponudb.

```
list(zip(cene, ponudb))
```

```
[(30, 4), (33, 3), (40, 4), (9, 1), (31, 4)]
```

Potem smo preprosto pogledali vsoto prvih elementov parov (`cena`) za vse tiste pare, pri katerih je drugi element (število ponudb) sod.

Toliko je porabila Berta, Ana pa toliko, kolikor je vsota cen izdelkov z lihim številom ponudb.

```
sum(cena for cena, ponudb_zanj in zip(cene, ponudb) if ponudb_zanj % 2 == 0)
```

```
101
```